|  |
| --- |
| **HTML** |

Enforce these, or your own, agreed upon guidelines at all times. Small or large, call out what's incorrect. For additions or contributions to this Code Guide, please [open an issue on GitHub](https://github.com/bwagh4/html_css_standards).

(Every line of code should appear to be written by a single person, no matter the number of contributors.)

1. **Syntax**

* Use soft tabs with two spaces—they're the only way to guarantee code renders the same in any environment.
* Nested elements should be indented once (two spaces).
* Always use double quotes, never single quotes, on attributes.
* Don't include a trailing slash in self-closing elements—the [HTML5 spec](http://dev.w3.org/html5/spec-author-view/syntax.html#syntax-start-tag) says they're optional.
* Don’t omit optional closing tags (e.g. </li> or </body>).

<!DOCTYPE html>

<html>

<head>

<title>Page title</title>

</head>

<body>

<img src="CDN/images/company-logo.png" alt="Company">

<h1 class="hello-world">Hello, world!</h1>

</body>

</html>

### 2) HTML5 doctype

Enforce standards mode and more consistent rendering in every browser possible with this simple doctype at the beginning of every HTML page.

<!DOCTYPE html>

<html>

<head>

</head>

</html>

### 3) Language attribute

From the HTML5 spec:

*Authors are encouraged to specify a lang attribute on the root html element, giving the document's language. This aids speech synthesis tools to determine what pronunciations to use, translation tools to determine what rules to use, and so forth.*

Head to Sitepoint for a [list of language codes](http://reference.sitepoint.com/html/lang-codes).

<html lang="en-us">

<!-- ... -->

</html>

### 4) IE compatibility mode

Internet Explorer supports the use of a document compatibility <meta> tag to specify what version of IE the page should be rendered as. Unless circumstances require otherwise, it's most useful to instruct IE to use the latest supported mode with **edge mode**.

For more information,

* [read this awesome Stack Overflow article](https://stackoverflow.com/questions/45507463/scroll-bar-hiding-element-in-internet-explore-11).
* [read this awesome Stack Overflow article](https://github.com/bwagh4/html_css_standards).

<meta http-equiv="X-UA-Compatible" content="IE=Edge">

### 5) Character encoding

Quickly and easily ensure proper rendering of your content by declaring an explicit character encoding. When doing so, you may avoid using character entities in your HTML, provided their encoding matches that of the document (generally UTF-8).

<head>

<meta charset="UTF-8">

</head>

**6) CSS and JavaScript includes**

Per HTML5 spec, typically there is no need to specify a type when including CSS and JavaScript files as text/css and text/javascript are their respective defaults.

HTML5 spec links

* Using link
* Using style
* Using script

<!-- External CSS -->

<link rel="stylesheet" href="code-guide.css">

<!-- In-document CSS -->

<style>

/\* ... \*/

</style>

<!-- JavaScript -->

<script src="code-guide.js"></script>

### 7) Practicality over purity

Strive to maintain HTML standards and semantics, but not at the expense of practicality. Use the least amount of markup with the fewest intricacies whenever possible.

**8) Attribute order**

HTML attributes should come in this particular order for easier reading of code.

* class
* id, name
* data-\*
* src, for, type, href, value
* title, alt
* role, aria-\*

Classes make for great reusable components, so they come first. Ids are more specific and should be used sparingly (e.g., for in-page bookmarks), so they come second.

<a class="..." id="..." data-toggle="modal" href="#">

Example link

</a>

<input class="form-control" type="text">

<img src="..." alt="...">

### 9) Boolean attributes

A boolean attribute is one that needs no declared value. XHTML required you to declare a value, but HTML5 has no such requirement.

For further reading, consult the [WhatWG section on boolean attributes](http://www.whatwg.org/specs/web-apps/current-work/multipage/common-microsyntaxes.html" \l "boolean-attributes):

*The presence of a boolean attribute on an element represents the true value, and the absence of the attribute represents the false value.*

If you must include the attribute's value, and **you don't need to**, follow this WhatWG guideline:

*If the attribute is present, its value must either be the empty string or [...] the attribute's canonical name, with no leading or trailing whitespace.*

**In short, don't add a value.**

<input type="text" disabled>

<input type="checkbox" value="1" checked>

<select>

<option value="1" selected>1</option>

</select>

### 10) Reducing markup

Whenever possible, avoid superfluous parent elements when writing HTML. Many times this requires iteration and refactoring, but produces less HTML. Take the following example:

<!-- Not so great -->

<span class="avatar">

<img src="...">

</span>

<!-- Better -->

<img class="avatar" src="...">

|  |
| --- |
| **CSS** |

1. **Syntax**

* Use soft tabs with two spaces—they're the only way to guarantee code renders the same in any environment.
* When grouping selectors, keep individual selectors to a single line.
* Include one space before the opening brace of declaration blocks for legibility.
* Place closing braces of declaration blocks on a new line.
* Include one space after : for each declaration.
* Each declaration should appear on its own line for more accurate error reporting.
* End all declarations with a semi-colon. The last declaration's is optional, but your code is more error prone without it.
* Comma-separated property values should include a space after each comma (e.g., box-shadow).
* Don't include spaces after commas *within* rgb(), rgba(), hsl(), hsla(), or rect()values. This helps differentiate multiple color values (comma, no space) from multiple property values (comma with space).
* Don't prefix property values or color parameters with a leading zero (e.g., .5 instead of 0.5 and -.5px instead of -0.5px).
* Lowercase all hex values, e.g., #fff. Lowercase letters are much easier to discern when scanning a document as they tend to have more unique shapes.
* Use shorthand hex values where available, e.g., #fff instead of #ffffff.
* Quote attribute values in selectors, e.g., input[type="text"]. [They’re only optional in some cases](http://mathiasbynens.be/notes/unquoted-attribute-values#css), and it’s a good practice for consistency.
* Avoid specifying units for zero values, e.g., margin: 0; instead of margin: 0px;.

Questions on the terms used here? See the [syntax section of the Cascading Style Sheets article](http://en.wikipedia.org/wiki/Cascading_Style_Sheets#Syntax) on Wikipedia.

/\* Bad CSS \*/

.selector, .selector-secondary, .selector[type=text] {

padding:15px;

margin:0px 0px 15px;

background-color:rgba(0, 0, 0, 0.5);

box-shadow:0px 1px 2px #CCC,inset 0 1px 0 #FFFFFF

}

/\* Good CSS \*/

.selector,

.selector-secondary,

.selector[type="text"] {

padding: 15px;

margin-bottom: 15px;

background-color: rgba(0,0,0,.5);

box-shadow: 0 1px 2px #ccc, inset 0 1px 0 #fff;

}

1. **Declaration order**

Related property declarations should be grouped together following the order:

1. Positioning
2. Box model
3. Typographic
4. Visual

Positioning comes first because it can remove an element from the normal flow of the document and override box model related styles. The box model comes next as it dictates a component's dimensions and placement.

Everything else takes place *inside* the component or without impacting the previous two sections, and thus they come last.

For a complete list of properties and their order, please see [Recess](http://twitter.github.com/recess).

.declaration-order {

/\* Positioning \*/

position: absolute;

top: 0;

right: 0;

bottom: 0;

left: 0;

z-index: 100;

/\* Box-model \*/

display: block;

float: right;

width: 100px;

height: 100px;

/\* Typography \*/

font: normal 13px "Helvetica Neue", sans-serif;

line-height: 1.5;

color: #333;

text-align: center;

/\* Visual \*/

background-color: #f5f5f5;

border: 1px solid #e5e5e5;

border-radius: 3px;

/\* Misc \*/

opacity: 1;

}

**3) Don't use** @import

Compared to <link>s, @import is slower, adds extra page requests, and can cause other unforeseen problems. Avoid them and instead opt for an alternate approach:

* Use multiple <link> elements
* Compile your CSS with a preprocessor like Sass or Less into a single file
* Concatenate your CSS files with features provided in Rails, Jekyll, and other environments

<!-- Use link elements -->

<link rel="stylesheet" href="core.css">

<!-- Avoid @imports -->

<style>

@import url("more.css");

</style>

### Media query placement

Place media queries as close to their relevant rule sets whenever possible. Don't bundle them all in a separate stylesheet or at the end of the document. Doing so only makes it easier for folks to miss them in the future. Here's a typical setup.

.element { ... }

.element-avatar { ... }

.element-selected { ... }

@media (min-width: 480px) {

.element { ...}

.element-avatar { ... }

.element-selected { ... }

}

### Prefixed properties

When using vendor prefixed properties, indent each property such that the declaration's value lines up vertically for easy multi-line editing.

In Textmate, use **Text → Edit Each Line in Selection** (⌃⌘A). In Sublime Text 2, use **Selection → Add Previous Line** (⌃⇧↑) and **Selection → Add Next Line** (⌃⇧↓).

/\* Prefixed properties \*/

.selector {

-webkit-box-shadow: 0 1px 2px rgba(0,0,0,.15);

box-shadow: 0 1px 2px rgba(0,0,0,.15);

}

### Single declarations

In instances where a rule set includes **only one declaration**, consider removing line breaks for readability and faster editing. Any rule set with multiple declarations should be split to separate lines.

The key factor here is error detection—e.g., a CSS validator stating you have a syntax error on Line 183. With a single declaration, there's no missing it. With multiple declarations, separate lines is a must for your sanity.

/\* Single declarations on one line \*/

.span1 { width: 60px; }

.span2 { width: 140px; }

.span3 { width: 220px; }

/\* Multiple declarations, one per line \*/

.sprite {

display: inline-block;

width: 16px;

height: 15px;

background-image: url(../img/sprite.png);

}

.icon { background-position: 0 0; }

.icon-home { background-position: 0 -20px; }

.icon-account { background-position: 0 -40px; }

1. **Shorthand notation**

Strive to limit use of shorthand declarations to instances where you must explicitly set all the available values. Common overused shorthand properties include:

* padding
* margin
* font
* background
* border
* border-radius

Often times we don't need to set all the values a shorthand property represents. For example, HTML headings only set top and bottom margin, so when necessary, only override those two values. Excessive use of shorthand properties often leads to sloppier code with unnecessary overrides and unintended side effects.

The Mozilla Developer Network has a great article on [shorthand properties](https://developer.mozilla.org/en-US/docs/Web/CSS/Shorthand_properties) for those unfamiliar with notation and behavior.

/\* Bad example \*/

.element {

margin: 0 0 10px;

background: red;

background: url("image.jpg");

border-radius: 3px 3px 0 0;

}

/\* Good example \*/

.element {

margin-bottom: 10px;

background-color: red;

background-image: url("image.jpg");

border-top-left-radius: 3px;

border-top-right-radius: 3px;

}

1. **Nesting in Less and Sass**

Avoid unnecessary nesting. Just because you can nest, doesn't mean you always should. Consider nesting only if you must scope styles to a parent and if there are multiple elements to be nested.

Additional reading:

* [Nesting in Sass and Less](http://markdotto.com/2015/07/20/css-nesting/)

// Without nesting

.table > thead > tr > th { … }

.table > thead > tr > td { … }

// With nesting

.table > thead > tr {

> th { … }

> td { … }

}

### Comments

Code is written and maintained by people. Ensure your code is descriptive, well commented, and approachable by others. Great code comments convey context or purpose. Do not simply reiterate a component or class name.

Be sure to write in complete sentences for larger comments and succinct phrases for general notes.

/\* Bad example \*/

/\* Modal header \*/

.modal-header {

...

}

/\* Good example \*/

/\* Wrapping element for .modal-title and .modal-close \*/

.modal-header {

...

}

1. **Class names**

* Keep classes lowercase and use dashes (not underscores or camelCase). Dashes serve as natural breaks in related class (e.g., .btn and .btn-danger).
* Avoid excessive and arbitrary shorthand notation. .btn is useful for *button*, but .s doesn't mean anything.
* Keep classes as short and succinct as possible.
* Use meaningful names; use structural or purposeful names over presentational.
* Prefix classes based on the closest parent or base class.
* Use .js-\* classes to denote behavior (as opposed to style), but keep these classes out of your CSS.

It's also useful to apply many of these same rules when creating Sass and Less variable names.

/\* Bad example \*/

.t { ... }

.red { ... }

.header { ... }

/\* Good example \*/

.tweet { ... }

.important { ... }

.tweet-header { ... }

1. **Selectors**

* Use classes over generic element tag for optimum rendering performance.
* Avoid using several attribute selectors (e.g., [class^="..."]) on commonly occuring components. Browser performance is known to be impacted by these.
* Keep selectors short and strive to limit the number of elements in each selector to three.
* Scope classes to the closest parent **only** when necessary (e.g., when not using prefixed classes).

/\* Bad example \*/

span { ... }

.page-container #stream .stream-item .tweet .tweet-header .username { ... }

.avatar { ... }

/\* Good example \*/

.avatar { ... }

.tweet-header .username { ... }

.tweet .avatar { ... }